CS 2451
Database Systems:
Relational Algebra &
Relational Calculus

http://www.seas.gwu.edu/~bhagiweb/cs2541

Spring 2020

Instructor: Dr. Bhagi Narahari

This week…

- Recap: Relational Algebra (RA)….a formal query language
  - SQL based off RA: "direct" translation between SQL and RA
- Relational Calculus (RC)…today
  - Set theory and predicate logic formulae
  - RA and RC exercises
- Intro to SQL – Data Definition Language component
  - Define schemas and populate tables: Use MySQL
  - Next class/lab
- In-Class exercises…work on team at table: turn in worksheet for grading

Review: Relational Algebra Operators

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in relation 1, but not in relation 2.
  - *Union* ($\cup$) Tuples in relation. 1 or in relation. 2.
  - *Rename*. $\rho_{X(C,D)}$ **(R (A,B))**
  - Create a "copy" of relation R with name X
- Additional operations:
  - Intersection, *join*, assignment, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, *operations can be composed*! (Algebra is "closed".)
- Relational algebra operates on relation
  - which are sets, therefore no duplicates!

How to write a RA query ?

- Find out which tables you need to access
  - Compute $\times$ of these tables
- What are the conditions/predicates you need to apply ?
  - Determines what select σ operators you need to apply
- What attributes/columns are needed in result
  - Determines what project π operators you need

Project (Select (Product))

$\pi_{(\text{attribute list})} (\sigma_{(\text{predicate conditions})} (R_1 \times R_2 \ldots))$

- Predicate conditions should include the join conditions

Modifying the Database

- Need to insert, delete, update tuples in the database
- What is insert ?
  - Add a new tuple to existing set = Union
- What is delete ?
  - Remove a tuple from existing set = Set difference
- How to update attribute to new value ?
  - Need new operator: δ

Modifying Database

- Delete all course enrollments of student with sid=3
  - Takes $\leftarrow$ Takes – (tuples of sid=3)
  - Takes $\leftarrow$ Takes – ($\sigma_{(sid=3)}$(Takes))
- Insert new student tuple (5, Kevin)
  - Student $\leftarrow$ Student $\cup$ (5, Kevin)
- Update: $\delta_{A \leftarrow E}(R)$
  - Update attribute A to E for tuples in relation R
  - $\delta_{exp\text{-}grade \leftarrow 'A'}$(Takes) : updates grade
  - Can also specify selection condition on Takest

    Update grade only for student with sid=5 and cid=500

Views: Important Concept

- Relational Model, using SQL, allows definition of a view
  - View is a virtual relation
  - Executed each time it is referenced
  - More when we get to SQL

Next….

- Relational Calculus
- Writing queries using RA and Rel. Calc.

Relational Calculus: An Equivalent, But Very Different, Formalism

- Codd invented a relational calculus that he proved was equivalent in expressiveness
  - Based on a subset of first-order logic – declarative, without an implicit order of evaluation
    - Tuple relational calculus
    - Domain relational calculus
  - More convenient for describing certain things, and for certain kinds of manipulations
- The DBMS uses the relational algebra internally, but query languages (e.g., SQL) use concepts from the relational calculus

Relational Calculus

- Comes in two flavors: *Tuple relational calculus* (TRC) and *Domain relational calculus* (DRC).
- Calculus has *variables, constants, comparison ops*, *logical connectives* and *quantifiers*.
  - *TRC*: Variables range over (i.e., get bound to) *tuples*.
  - *DRC*: Variables range over *domain elements* (= field values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*.
- **An *answer tuple* is an assignment of values to variables that make the formula evaluate to *true*.**

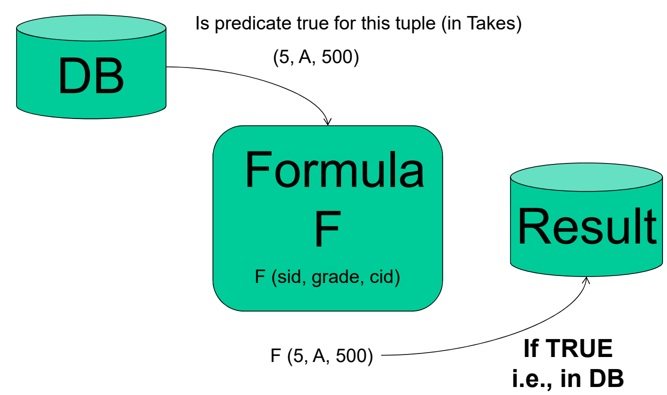
Tuple Relational Calculus (RC)

- A tuple variable is a variable whose values can be tuples from a relational schema
- Formula/Query in RC is expressed as:

  { t | P(t) }
  - t is a tuple variable
  - P(t) is property of tuple t; it is a predicate formula that describes properties of the tuple variable
    - Thereby defining the possible values of t
  - Result is set of all tuples where predicate P is true
- *Formula* is recursively defined:
  - start with simple *atomic formulas*

    (get tuples from relations or make comparisons of values)
  - build bigger formulas using *logical connectives*.

Relational Calculus



Data Instance for Mini-Banner Example

STUDENT

sid	name
1	Jill
2	Matt
3	Jack
4	Maury

Takes

sid	exp-grade	cid
1	A	550-0103
1	A	700-1003
3	A	700-1003
3	C	500-0103
4	C	500-0103

COURSE

cid	subj	sem
550-0103	DB	F03
700-1003	Math	S03
501-0103	Arch	F03

PROFESSOR

fid	name
1	Narahari
2	Youssef
8	Choi

Teaches

fid	cid
1	550-0103
2	700-1003
8	501-0103

- Find (tuples) students with name “Jill”
- “syntax” is set theory notations

{ t | t ∈ Students ∧ t[name] = ‘Jill’}

Tuple Relational Calculus (RC) – more syntax

- A tuple variable is a variable whose values can be tuples of a relational schema
- Formula/Query in RC is expressed as:
  { (t[att1], t[att2],..) | R(t) AND P(t) }
  - t is a tuple variable in relation R
  - t[att1] is value of attribute 1 in tuple t
  - P(t) is property of tuple t; it is a predicate formula that describes properties of the tuple variable
    - Thereby defining the possible values of t
  - Result is set of all tuples for which predicate P is true

Alternate syntax

- Find all students with name Kevin

{ (t.sid, t.name) | Student(t) ∧ t[name] = “Kevin”}

Note: We will use set theory notations (to continue with what you learnt in CS1311!)

Domain Relational Calculus (DRC)

Queries have form:

$$\{\underbrace{\langle x_1, x_2, \ldots, x_n\rangle}_{\text{domain variables}} \mid p\}$$

predicate

- Predicate: boolean expression over $x_1, x_2, \ldots, x_n$
- *Answer* includes all tuples that make the *formula true*.
- The variables come from the domain of the attributes in the relation schema
  - in contrast to the tuple calculus where variables are tuples
- **we will be working with tuple relational calculus (TRC)**

Domain Relational Calculus

- Define domain of each attribute in result set and the type
- Find name, sid for students whose name is Lily

{ <b,a> | <b,a> ∊ Student ∧ a='Lily' }

answer tuples: what are all the values that b can take on?

only values where the name field = 'Lily'

More definitions of formulas in RC

- This is nothing but "discrete math on steroids"!!

RC Formulas

- *Atomic formula:*
  - $t \in Rname$, or X *op* Y, or X *op* constant

  op is one of $<, >, =, \leq, \geq, \neq$
- *Formula:* an atomic formula, or
  - $\neg p, p \wedge q, p \vee q$, where p and q are formulas, or
  - $\exists X(p(X))$, where variable X is *free* in p(X), or
  - $\forall X(p(X))$ where variable X is *free* in p(X)
- The use of quantifiers $\exists X$ and $\forall X$ is said to *bind* X.
  - A variable that is not bound is free.

Expressions and Formulas in RC

- Truth value of an atomic formula (atom) evaluates to either TRUE or FALSE
- Formula is made up of one or more atoms connected via logical operations AND, OR, NOT…

Existential and Universal Quantifiers

- Two special symbols can appear in formulas:
  - $\forall\, t$: universal quantifier
  - $\exists\, t$: existential quantifier
- Informally: a tuple is bound if it is quantified- it appears in an universal or existential clause, otherwise it is free
- If F is a formula, then so are $(\exists\, t)(F)$ and $(\forall t)(F)$
  - The formula $(\exists\, t)(F)$ is true if the formula F evaluates to true for some (at least one) tuple assigned to free occurences of t in F; otherwise it is false.
  - The formula $(\forall t)(F)$ is true if the formula F evaluates to true for every tuple (in the universe) assigned to free occurrences of t in F; otherwise it is false.
  - ∀ is called universal "for all" quantifier because every tuple in the universe of tuples must make F true to make the formula true
  - ∃ is called existential or "there exists" quantifier because any tuple that exists may make F true to make formula true

More Complex Predicates in Relational Calculus

Starting with these atomic predicates, build up new predicates by the following rules:

- Logical connectives: If p and q are predicates, then so are

  $p \wedge q$, $p \vee q$, $\neg p$, and $p \Rightarrow q$
  - $(x>2) \wedge (x<4)$ = ? (True or false)
  - $(x>2) \wedge \neg(x>0)$ = ?
- Existential quantification: If p is a predicate, then so is $\exists x.p$
- Universal quantification: If p is a predicate, then so is $\forall x.p$

Review….Existential and Universal Quantifiers

- Existential Quantifiers: for natural numbers
  $\exists x.\ ((x>2) \wedge (x<4))$ evaluates to ?
- Universal Quantifier- (for natural numbers) $\forall x.(x>2)$ evaluates to ?
- For domain of natural numbers,
  $\forall x\ \exists y\ (y>x)$ evaluates to ?

Logical Equivalences

- Recall from discrete math cs1311
- There are two logical equivalences that are heavily used:
  - $p \Rightarrow q \equiv \neg p \vee q$
    (Whenever p is true, q must also be true.)
  - $\forall x.\ p(x) \equiv \neg\exists x.\ \neg p(x)$
    (p is true for all x)
- The second can be a lot easier to check!
- Example:
  - The highest course number offered

Free and Bound Variables

- The use of quantifiers $\forall$ or $\exists$ in a formula is said to bind the variables
  - A variable *v* is bound in a predicate *p* when *p* is of the form $\forall v\ldots$ or $\exists v\ldots$
- A variable occurs free in *p* if it occurs in a position where it is not bound by an enclosing $\forall$ or $\exists$
- Examples:
  - x is free in $x > 2$ y is free and x is bound in $\exists x.\ x > y$
- Important restriction: the variable *t* that appear to the left of `|' must be the *only* free variables in the formula P(*t*).
  - All other tuple variables must be bound using quantifier
- Implication: the values that the free variables can legally take on are the results of the query!

Safety of Operators

- Query of the form $\exists\ t \in R\ (Q(t))$
  - *There exists tuple t in set/relation R such that predicate Q is true*
- Safety of Expressions
  - What about $\{ t \mid \neg(t \in \text{Student}) \}$

    Infinitely many tuples outside loan relation

Safety of Expressions

- A query is safe if no matter how we instantiate the relations, it always produces a finite answer
  - Domain independent: answer is the same regardless of the domain in which it is evaluated
  - Unfortunately, both this definition of safety and domain independence are semantic conditions, and are undecidable
- There are syntactic conditions that are used to guarantee “safe” formulas
  - One solution: For each tuple relational formula P, define domain Dom(P) which is set of all values referenced by P
  - The formulas that are expressible in real query languages based on relational calculus are all “safe”

    Many DB languages include additional features, like recursion, that must be restricted in certain ways to guarantee termination and consistent answers

Data Instance for Mini-Banner Example

STUDENT

sid	name
1	Jill
2	Matt
3	Jack
4	Maury

Takes

sid	exp-grade	cid
1	A	550-0103
1	A	700-1003
3	A	700-1003
3	C	500-0103
4	C	500-0103

COURSE

cid	subj	sem
550-0103	DB	F03
700-1003	Math	S03
501-0103	Arch	F03

PROFESSOR

fid	name
1	Narahari
2	Youssef
8	Choi

Teaches

fid	cid
1	550-0103
2	700-1003
8	501-0103

Examples: Relational Calculus

- Find sid, grade, and course ID for grades of A
  - What is the "type" of the elements in the result, i.e., where do they come from ?
  - What is the property of the elements ?
- $\{ t \mid (t \in \text{Takes}) \wedge (t[\text{exp-grade}] = \text{'A'}) \}$
  - Type of tuple *t* is Takes (since it is an element of Takes)
  - Property is that value of exp-grade attribute in the tuple must be equal to A

Domain Relational Calculus

- Define domain of each attribute in result set and the type
- Find sid, grade, cid for grades=A
- $\{ <a,b,c> \mid <a,b,c> \in Takes \wedge \quad b = \text{'A'} \}$
  - Domain of each attr in result is defined by <a,b,c> is an element in Takes

Relational calculus – projections: free variables

- Two approaches depending on use of set notation or not..
- What if the type has to be inferred ?
- Find only Student ID attribute in the previous example
  - This type has to be inferred by the query
  - Tuples on ID, for which there is a tuple in Takes with same sid and exp-grade= 'B'.
- $\{t \mid \exists s \in \text{Takes} (s[sid]=t[sid] \wedge s[\text{exp-grade}]= \text{'B'} \}$
  - 'schema' of *t* can be deduced, from query, as containing an attribute sid
  - No other attribute is defined for *t*

    Therefore the 'type' of t is [sid] (a single attribute)
- Note use of existential quantifier…
  - s is bound variable
  - t is free variable….result of query is values that free variable can take to make the predicate true

Using named field notation…easier

{*t.sid* | ∃ s∈ Takes (s[sid]=t[sid] ∧s[exp-grade]='B') }

Cross Products in TRC

- Find names of students who have a grade of B in some course
  - for sid with grades of B we had
  - *{t | ∃ s∈ Takes(s[sid]=t[sid] ∧s[grade]= 'B') }*
- *How about name? It exists in Student relation ?*
  - *For tuple c ∈ Student what property does c have?*
    Use tuple c to "join" the two relations
- The sid in tuple *c in* Student relation is same as sid in tuple *s* in Takes relation
- Free variable ?
  - Result tuples *t* must have only name
  - So this is the only attribute for which *t* is defined in the predicate condition

Cross products in TRC

{*t* | ∃ s∈ Takes, ∃ c∈ Student (t[name]=c[name] ∧ s[grade]= 'B' ∧ c[sid]=s[sid]) }

- *t* is free variable
- Its 'type' is single attribute/column called name
- The values *t* can take on are restricted to c[name] where the student appears in the Takes relation and has grade equal to B

Summary: Relational Model, Formal Query languages

- *Relational Completeness*: Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus.
- Relational calculus is non-operational
  - users define queries in terms of what they want, not in terms of how to compute it.
- Algebra and safe calculus have same expressive power, leading to the notion of relational completeness.

Recap: How to write a RA query ?

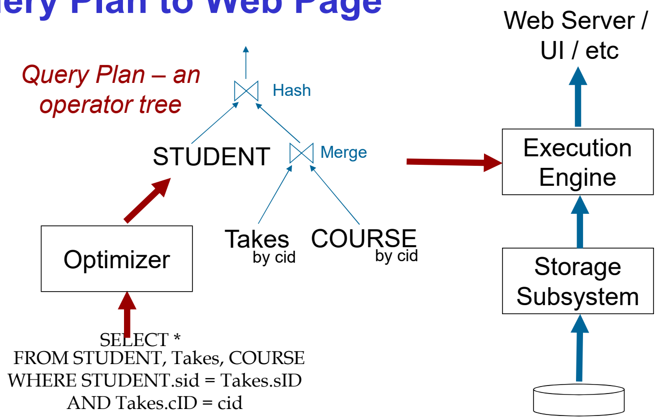
- Find out which tables you need to access
  - Compute × of these tables
- What are the conditions/predicates you need to apply ?
  - Determines what select σ operators you need to apply
- What attributes/columns are needed in result
  - Determines what project π operators you need

Why Formal languages ? Example: Optimization Is Based on Algebraic Equivalences

- Relational algebra has laws of commutativity, associativity, etc. that imply certain expressions are equivalent in semantics
- They may be different in cost of evaluation!
  - $\sigma_{(P1 \wedge P2)}(R) = \sigma_{P1}(\sigma_{P2}(R))$
  - $(R1 \bowtie R2) = (R2 \bowtie R1)$
  - $(R1 \bowtie R2) \bowtie R3 = R1 \bowtie (R2 \bowtie R3)$

❖ Query optimization finds the most efficient representation to evaluate (or one that's not bad)

The Big Picture: SQL to Algebra to Query Plan to Web Page



Time to practice…In-class exercises

- Work at your table…..use the whiteboard
  - Raise your hand if you have a question
- Write down the answers and hand them in end of class
  - Write your names